(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 927 936 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
07.07.1999 Bulletin 1999/27

(51) Int Cl.6: G06F 12/06, G06F 15/76,
G06F 13/16

(21) Application number: 98310349.0

(22) Date of filing: 16.12.1998

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 31.12.1997 US 70218 P

(71) Applicant: TEXAS INSTRUMENTS INC.
Dallas, Texas 75243 (US)

(72) Inventors:
- Baltz, Philip K.
  Houston, TX 77083 (US)
- Simar, Ray L Jr.
  Richmond, TX 77469 (US)

(74) Representative: Potter, Julian Mark et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) A microprocessor with configurable on-chip memory

(57) A processor structure and method of operation are disclosed that comprise a user-configurable on-chip program memory. The memory comprises an on-chip memory 31 and a program memory controller 30 that reconfigures memory 31 in response to control values that may be modified by CPU core 20 under program control. In one mode, memory 31 may be mapped into internal address space. In other modes, memory 31 may be configured as an on-chip cache. In conjunction with the cache configuration, the program memory controller may comprise a tag RAM that is initialized upon a transition to cache mode. Program memory controller 30 handles memory mode transitions and data requests; CPU core 20 preferably requests stored instructions from controller 30 in a uniform fashion regardless of memory mode.

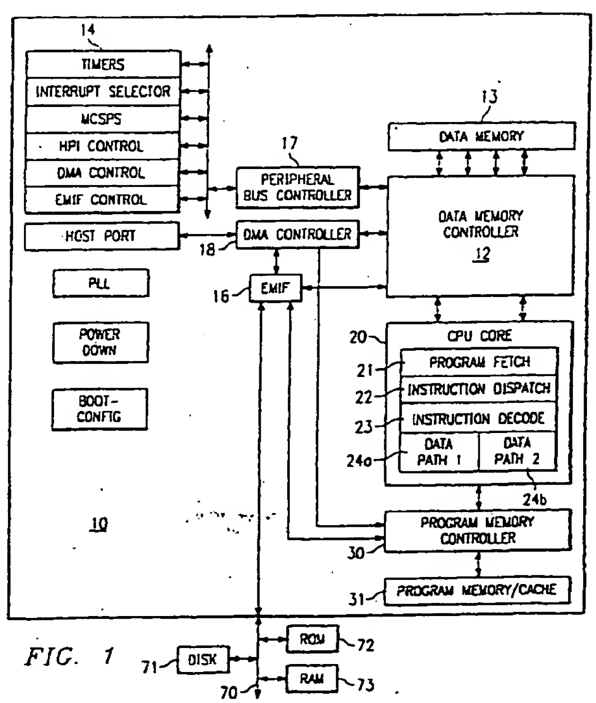


FIG. 1

EP 0 927 936 A2

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION

[0001] The present invention pertains generally to microprocessor architectures, and pertains more particularly to microprocessors having on-chip program memory capability.

BACKGROUND OF THE INVENTION

[0002] A microprocessor is a circuit that combines the instruction-handling, arithmetic, and logical operations of a computer on a single chip. A digital signal processor (DSP) is a microprocessor optimized to handle large volumes of data efficiently. Such processors are central to the operation of many of today's electronic products, such as high-speed modems, high-density disk drives, digital cellular phones, and complex automotive systems, and will enable a wide variety of other digital systems in the future. The demands placed upon DSPs in these environments continue to grow as consumers seek increased performance from their digital products.

[0003] Designers have succeeded in increasing the performance of DSPs and microprocessors in general by increasing clock speeds, by removing architectural bottlenecks in circuit designs, by incorporating multiple execution units on a single processor circuit, and by developing optimizing compilers that schedule operations to be executed by the processor in an efficient manner. As further increases in clock frequency become more difficult to achieve, designers have embraced the multiple execution unit processor as a means of achieving enhanced DSP performance. For example, Figure 2 shows a block diagram of the CPU data paths of a DSP having eight execution units, L1, S1, M1, D1, L2, S2, M2, and D2. These execution units operate in parallel to perform multiple operations, such as addition, multiplication, addressing, logic functions, and data storage and retrieval, simultaneously.

[0004] Theoretically, the performance of a multiple execution unit processor is proportional to the number of execution units available. However, utilization of this performance advantage depends on the efficient scheduling of operations so that most of the execution units have a task to perform each clock cycle. Efficient scheduling is particularly important for looped instructions, since in a typical runtime application the processor will spend the majority of its time in loop execution.

[0005] Unfortunately, the inclusion of multiple execution units also creates new architectural bottlenecks. Increased functionality translates into longer instructions, such as may be found in very long instruction word (VLIW) architectures. For example, the eight-execution unit VLIW processor described above may require a 256-bit instruction every clock cycle in order to perform tasks on all execution units. As it is generally neither practical nor desirable to provide, for example a 256-bit-wide parallel data path external to the processor merely for instruction retrieval, the data rate available for loading instructions may become the overall limiting factor in many applications. It is an object of the teachings disclosed herein to propose a solution to resolve this bottleneck.

SUMMARY OF THE INVENTION

[0006] Many high performance signal processors provide at least some program memory on-chip because of the delays associated in loading instructions from external memory. However, the area on a microprocessor allotted for on-chip memory is by necessity limited, and prior art on-chip memories provide no ability to reconfigure this limited and precious resource. The present teachings seeks to solve a heretofore unrecognized problem--given that the core functionality of some applications can be loaded on-chip to a sufficiently-sized memory, while the core functionality of others cannot, can an on-chip memory be designed to meet the needs of either type of application, without duplicating and possibly wasting resources? It has now been recognized that an on-chip memory that is configurable by the user, preferably in software, will provide the maximum flexibility for all applications. The present teachings disclose a microprocessor with an on-chip memory that may be configured at runtime to one of several memory modes as requested by an application.

[0007] In one aspect of the teachings, a microprocessor is disclosed that comprises a configurable on-chip memory. Preferably, the microprocessor further comprises a program memory controller that allows the current on-chip memory configuration to remain transparent to the microprocessor central processing unit (CPU) core during program memory operations. Preferably, the configurable on-chip memory may be configured as either memory-mapped or cache memory. The cache memory may preferably be further configured to operate in multiple modes, e.g., fully enabled, bypassed, or read-only.

[0008] In a second aspect of the teachings, the configurable on-chip memory may be reconfigured during microprocessor operation under software control. For instance, a configurable memory may be booted in one mode, and subsequently switched, once or multiple times, to other modes, by software commands executed by the CPU of the microprocessor. Such software commands preferably alter the operation of the program memory controller and on-chip memory by changing a control signal on the microprocessor.

[0009] In yet another aspect of the teachings, the program memory controller (PMC) operates in either a memory-mapped mode or a cache mode to determine if requested addresses are on-chip memory addresses. The program memory controller preferably supplies requested fetch packets if on-chip, or halts the processor and loads requested fetch packets from off-chip. The

PMC checks for requests for memory mode transitions and initiates transitions when the CPU requests such.

[0010] In a further aspect of the teachings, a tag RAM is associated with cache memory operation. This tag RAM preferably operates in conjunction with the program memory controller, which determines if the fetch packet at the requested address is currently loaded into the cache. The program memory controller preferably has the capability to update the tag RAM when a fetch packet is loaded from off-chip. The program memory controller preferably also has the capability to re-initialize the tag RAM during microprocessor operation, e.g., due to a switch in memory configuration.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention will now be further described, by way of example, with reference to the accompanying drawings in which:

Figure 1 is a block diagram depicting the major functional blocks of a processor implementation .
Figure 2 is a block diagram illustrating a configuration of execution units and registers of a multiple-execution unit processor;
Figure 3 shows the arrangement of instructions in a fetch packet;
Figures 4a and 4b show maps of processor address space for two different memory mappings;
Figure 5 depicts instruction address partitioning for use as a cache address;
Figure 6 depicts the interface between the CPU core and the program memory controller;
Figure 7 illustrates the states and allowable state transitions for a program memory controller.
Figure 8 shows the configuration of a status register that may be used to control a configurable memory ; and
Figure 9 shows the registers and data paths of a program memory controller.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0012] Several illustrative embodiments are described herein . Although it is believed that the teachings disclosed herein may be readily adapted to virtually any CPU architecture, for illustrative purposes these embodiments are described with reference to a specific VLIW processor family, the Texas Instruments TMS320C6x. Those of ordinary skill in the pertinent art should comprehend the description below in sufficient detail to enable them to reproduce the invention; however, for specific data related to processor architecture, instruction set, and operation, the interested reader is referred to the Texas Instruments TMS320C62xx CPU and Instruction Set Reference Guide (1997) and the Texas Instruments TMS320C62xx Peripherals Reference Guide (1997), which are incorporated herein by reference.

[0013] Several definitions should also be useful to the reader. As used herein, an *instruction* is a function performable by an execution unit on a processor in one or more clock cycles. An *execute packet* is a set of one or more instructions that will be dispatched to the execution units during the same clock cycle. A *fetch packet* is a standard-sized block of instructions, comprising one or more execute packets, that is loaded into the CPU as a single unit.

[0014] A *memory-mapped* on-chip memory occupies a contiguous section of regularly addressable program memory. A *cache* on-chip memory contains a copy of instructions that also reside in external memory and that have been previously requested (usually those most recently requested) by the CPU. These do not necessarily represent a contiguous section of program memory, and are not generally explicitly addressable by the CPU.

[0015] The Texas Instruments TMS320C6x (C6x) processor family comprises several preferred embodiments . The C6x family includes both scalar and floating-point architectures. The CPU core of these processors contains eight execution units, each of which requires a 31-bit instruction. If all eight execution units of a processor are issued an instruction for a given clock cycle, the maximum instruction word length of 256 bits (8 31-bit instructions plus 8 bits indicating parallel sequencing) is required.

[0016] A block diagram of a C6x processor connected to several external data systems is shown in Figure 1. Processor 10 comprises a CPU core 20 in communication with program memory controller 30 and data memory controller 12. Other significant blocks of the processor include peripherals 14, a peripheral bus controller 17, and a DMA controller 18.

[0017] Processor 10 is configured such that CPU core 20 need not be concerned with whether data and instructions requested from memory controllers 12 and 30 actually reside on-chip or off-chip. If requested data resides on chip, controller 12 or 30 will retrieve the data from respective on-chip data memory 13 or program memory/cache 31. If the requested data does not reside on-chip, these units request the data from external memory interface (EMIF) 16. EMIF 16 communicates with external data bus 70, which may be connected to external data storage units such as a disk 71, ROM 72, or RAM 73. External data bus 70 is 32 bits wide.

[0018] CPU core 20 includes two generally similar data paths 24a and 24b, as shown in Figure 1 and detailed in Figure 2. The first path includes a shared multiport register file A and four execution units, including an arithmetic and load/store unit D1, an arithmetic and shifter unit S1, a multiplier M1, and an arithmetic unit L1. The second path includes register file B and execution units L2, S2, M2, and D2. Capability (although limited) exists for sharing data across these two data paths.

[0019] Because CPU core 20 contains eight execu-

tion units, instruction handling is an important function . Groups of instructions are requested by program fetch 21 and received from program memory controller 30 as fetch packets. Instruction dispatch 22 distributes instructions from fetch packets among the execution units as execute packets, and instruction decode 23 decodes the instructions.

[0020] In the preferred embodiment, a fetch packet has a fixed length of eight instructions, as shown in Figure 3. The execution grouping of the fetch packet is specified by the *p*-bit, bit zero, of each instruction. Fetch packets are eight-word aligned in program memory.

[0021] The *p*-bit controls the parallel execution of instructions. The p-bits are scanned from left to right (lower to higher address) by instruction dispatch 22. If the *p*-bit of instruction *i* is 1, then instruction *i*+1 is to be executed in parallel with instruction *i*, i.e., in the same execute packet. Thus an execute packet may contain from one to eight instructions, and a fetch packet may contain from one to eight execute packets, depending on the size of the execute packets. All instructions in an execute packet must utilize a unique execution unit. An execute packet also cannot cross an eight-word boundary. Thus, the last *p*-bit in a fetch packet is always set to 0, and each fetch packet starts with a new execute packet.

[0022] Because of this variable execute packet length and fixed fetch packet length, on-chip program memory 31 in the preferred embodiment is aligned by fetch packets. If an instruction that resides in the middle of a fetch packet is requested by the CPU, the entire fetch packet is retrieved, but all instructions at lower addresses are ignored (even if they would have otherwise operated in parallel with the requested instruction).

[0023] The physically addressable address space of the C6x processor is 4 Gbytes. On-chip program memory 31 has a size of 64K bytes. However, each instruction requires four bytes, and each fetch packet contains eight instructions, such that on-chip program memory 31 is arranged as 2K *frames,* each frame holding one fetch packet of 32 bytes, or 256 bits, in length. In memory map mode, the 64K bytes of on-chip memory may be selected to reside at a contiguous block of memory in address space starting at address 140 0000, as shown in Figure 4A, or at a starting address of 000 0000, as shown in Figure 4B.

[0024] In cache mode, the representative embodiments assume that instructions will occupy a maximum external address space of 64 Mbytes. Thus the cache in these embodiments ignores the top six bits of an address in cache mode, as shown in Figure 5. The cache also ignores the bottom five bits of an address, as the cache stores only fetch-packet-aligned (i.e. 32-byte-aligned) data. Bits 5 to 25 of an instruction address are the only bits used to map external address space into cache locations.

[0025] As shown in Figure 5, bits 5 to 25 are divided within the PMC into a ten-bit tag (bits 16-25) and an eleven-bit block offset (bits 5-15). The program memory controller 30 contains a tag RAM 32 (see Figure 9) that is capable of storing 2K tags, one for each frame in memory 31, in order to track the contents of the cache. The eleven-bit block offset is used both as an address for the appropriate tag within tag RAM 32 and as an address for the appropriate frame within memory 31. Each eleven-bit location within tag RAM 32 contains a validity bit and a ten-bit tag. Although external addresses 64k apart map to the same location in the tag RAM, each external address maps to a unique combination of block offset and tag.

[0026] When the cache is initialized and enabled, the validity bit at each tag location is marked invalid. Then, as each new fetch packet is requested, its address is partitioned within PMC 30 into a compare tag and a block offset. The block offset is used to retrieve a tag from tag ram 32. If the tag validity bit is invalid, it is set and the compare tag is written into the tag RAM using the block address as an offset, and a cache miss is declared. If the tag validity bit of the retrieved tag is set, the retrieved tag is compared to the compare tag in tag comparator 34. If the two tags fail to match, a cache miss is declared and the compare tag is written into the tag RAM using the block address as an offset. If the two tags are identical, comparator 34 registers a cache hit and the tag ram is not modified.

[0027] If a cache hit occurs, the requested fetch packet is retrieved from on-chip memory 31 using the block offset as an address. With a cache miss, the requested fetch packet is retrieved by sending the external address to EMIF 16 for off-chip retrieval. As the instructions of the fetch packet are received from EMIF 16 they are written into on-chip memory 31 one 32-bit instruction at a time, using the block offset as an address. Once the entire fetch packet is received, it is sent to the CPU.

[0028] Although the cache is typically fully enabled during caching, several other cache modes are available to the user. Cache freeze mode operates similar to cache enable mode, except that the cache and tag ram are never updated. This mode is useful for protecting valuable cache contents, e.g., during interrupt service. Cache bypass mode causes a cache miss on every fetch, effectively removing on-chip memory 31 from service.

[0029] During processor operation, on-chip memory operations are preferably transparent to the CPU, such that program data requests and program data stores are handled in a uniform fashion. Referring now to Figure 6, the PMC and the CPU interface with a program address bus 44, a program data bus 43, and several control signals. The PROGRAM ADDRESS STROBE (PAS) signal is sent by the CPU when it places an instruction request on the program address bus. The PROGRAM DATA STROBE (PDS) signal is sent by the CPU when it needs program data (this typically occurs one to eight CPU cycles after the PAS signal is sent). The PROGRAM WRITE STROBE (PWS) signal is sent by the CPU when it desires to write data to program memory. The PMC

uses the RDY signal to acknowledge that it is supplying requested fetch packets as needed. The RDY signal is taken low to stall the CPU if the PMC cannot produce the program data when the PDS requests it. The RDY signal may also be taken low at other times, as described below.

[0030] Figure 7 illustrates the states and allowable state transitions for the program memory controller of the C6x processor embodiment. These states may be divided generally into three categories as shown: memory map states, cache states, and transition states. A description of each state and its corresponding state transition conditions follows.

[0031] Referring again to Figure 7, RESET PMC is the boot state of the PMC. The PMC typically stays in this state whenever the RESET pin of the processor is asserted. However, the PMC may transition to a BOOT LOAD state from RESET PMC if the DMA provides a request during RESET. During BOOT LOAD, the DMA may store data into the on-chip memory. Once the DMA request has been serviced in BOOT LOAD, the PMC transitions back to RESET PMC.

[0032] Upon release of RESET, the PMC transitions to memory map mode and the FETCH RUN state. FETCH RUN is the default state of the PMC in memory map mode. The PMC idles in this state until a request is received. If the CPU has requested a fetch packet by asserting PAS, the PMC determines if the address on bus 44 is an on-chip memory address. If the address is an on-chip address, the requested fetch packet is placed on the program data bus. If the address is an off-chip address, the PMC sends the address to the EMIF for program data retrieval.

[0033] The PMC transitions from FETCH RUN to FETCH STALL if the requested fetch packet has not been retrieved before the CPU indicates it needs the data by asserting PDS (typically one to eight clock cycles after the CPU asserts PAS). In FETCH STALL, the PMC halts the CPU by deasserting the RDY signal until the requested fetch packet has been received. Once the PMC retrieves the fetch packet, the PMC transitions back to FETCH RUN and RDY is reasserted.

[0034] The PMC may also transition from FETCH RUN to WRITE ON CHIP if a store program (STP) instruction is executed by the CPU. The STP instruction causes the CPU to assert PWS, indicating to the PMC that an instruction write is requested. In WRITE ON CHIP, the program address on address bus 44 is evaluated by the PMC; if it is a valid on-chip address, the instruction on program data bus 43 is written into on-chip memory 31 and the PMC transitions back to FETCH RUN. If the address is an off-chip address, the PMC transitions to WRITE OFF CHIP. In either case, WRITE ON CHIP is a one-cycle state. RDY is deasserted in this state.

[0035] The WRITE OFF CHIP state is only entered from WRITE ON CHIP, and RDY remains deasserted in this state. WRITE OFF CHIP passes the instruction address and data to the EMIF for writing. The PMC remains in this state until the EMIF has written the data, and then transitions back to FETCH RUN.

[0036] The final memory mode state is DMA REQUEST. The DMA can write to on-chip memory during this one-cycle state. However, the CPU is given priority over the DMA, and no transition from FETCH RUN to DMA REQUEST will occur as long as the CPU has pending requests. Note also that no corresponding state exists for cache operation--as the cache stores a copy of off-chip memory, the results of a write only to on-chip cache would be unstable. Thus, DMA requests in cache mode are ignored. As an alternative, the DMA request could be handled similar to STP requests in cache mode (see the CACHE WRITE state below).

[0037] The PMC has a separate set of states for memory and cache modes, although functional similarities exist between the two modes. The resting cache mode state is STROBE WAIT RUN; the PMC returns to this state when there are no pending fetches, and remains in this state until the CPU asserts PAS or PWS.

[0038] When the CPU asserts PAS, the PMC transitions to HIT RUN. In this state, the PMC determines if the cache contains a valid replica of the requested fetch packet. If it does, a cache hit is declared and the packet is returned from the cache, and the PMC transitions back to STROBE WAIT RUN unless another request is pending. If the requested fetch packet is not in the cache, the PMC declares a miss and transitions to MISS RUN. RDY remains asserted in HIT RUN.

[0039] In MISS RUN, RDY remains asserted as the PMC fetches the requested packet from off-chip via the EMIF. In this state, if the cache is fully enabled the tag RAM will be updated and the packet will be written into the corresponding cache location as it is received from off-chip. The PMC remains in MISS RUN until the entire packet is fetched, unless the CPU requests the fetch packet data before the fetch is completed, in which case a transition to MISS STALL occurs. Once the fetch is completed, the PMC may transition back to STROBE WAIT RUN if no further requests are pending, to HIT RUN if an in-cache request is pending, or remain in MISS RUN if an off-chip request is pending.

[0040] If the CPU requests off-chip data before it has been completely retrieved, the PMC transitions to MISS STALL, deasserts RDY, and stalls the CPU until the fetch has completed. Once the off-chip fetch is completed, the PMC transitions to MISS RUN if an additional off-chip request is pending; otherwise, it transitions to HIT RUN.

[0041] The PMC may also transition from STROBE WAIT RUN, HIT RUN, or MISS STALL to CACHE WRITE if the CPU asserts the PWS signal (the transition occurs after pending fetch requests are completed). In CACHE WRITE, the CPU is stalled by deasserting RDY, and the data on program data bus 43 is written to the physical off-chip address appearing on program address bus 44. In this state, the tag associated with this

address is cleared in the tag RAM. One alternative to clearing the tag would be to update the tag RAM and on-chip memory after writing the new value into off-chip memory.

[0042] Although the C6x has been designed to always boot the on-chip memory in memory map mode, one of the key features of the disclosed teachings is the ability to reconfigure on-chip memory during processor operation. Although this could be done with an externally-supplied signal, in the preferred embodiment the CPU controls the mode of on-chip memory. As illustrated in Figure 8, the C6x CPU Control Status Register (CSR) contains a PCC field that indicates the desired program memory mode, and is observable by the program memory controller. In the C6x, the PCC is implemented as a three-bit field with four valid values (the other four are reserved for future implementation of additional modes). PCC value 000 represents memory mapped mode, and is the reset state. PCC value 010 represents cache enabled mode. PCC value 011 represents cache freeze mode, where cache contents are retained and readable, but off-chip reads do not affect the cache. And PCC value 100 represents cache bypass mode, which essentially bypasses on-chip memory and forces all reads to come from off-chip.

[0043] The user may select a PCC value that provides best performance for an application or portion of an application then executing on the processor. The user typically changes the PCC value by reading the CSR, modifying the PCC field, and writing the modified contents back into the CSR. From the standpoint of the PMC state machine, the most significant PCC events are transitions between the memory map state and one of the cache states.

[0044] While in memory map mode, the PMC checks the value of PCC in FETCH RUN and FETCH STALL states. If the PCC changes to a cache state, after the current fetch request is completed the PMC will transition to MEM TO CACHE. MEM TO CACHE stalls the CPU while it initializes tag RAM 32 by clearing the valid bit associated with each tag. Although different implementations are possible, the C6x clears the bits one tag per clock cycle. The PMC in the C6x remains in MEM TO CACHE for 2049 clock cycles, 2048 of these being required to clear the 2K tags in the tag RAM.

[0045] If no fetch requests were pending at the transition to MEM TO CACHE, the PMC transitions to STROBE WAIT RUN in cache mode after initializing the tag RAM. If a request was pending, the PMC transitions instead to MISS STALL.

[0046] The PMC performs a similar check of PCC in cache mode. However, it will not transition to memory map mode until a cache miss occurs, i.e., transitions to the CACHE TO MEM state occur from the MISS RUN and MISS STALL states. In CACHE TO MEM, the PMC stalls the CPU. CACHE TO MEM clears up any pending fetch requests and then transitions to FETCH RUN in memory map mode.

[0047] In this embodiment, the PMC takes no action with regard to the on-chip memory upon transition from cache to memory map mode. Thus the user is responsible for insuring that the memory-map contents are not used without proper initialization. Other embodiments of CACHE TO MEM are possible, such as one that fills on-chip memory from a specified location in off-chip memory before transitioning to memory-map mode.

[0048] The registers and data paths through the PMC are illustrated in Figure 9. Because the CPU core 20 is allowed to request a second fetch packet before it is ready to receive a first, two pipelined address registers 35 and 36 are used to handle multiple fetch requests. Likewise, both requests may be serviced (typically if both are on-chip) before CPU core 20 is ready for data, thus two pipelined data registers 37 and 38 are used to sequence retrieved data. Write data register 39 and write address register 40 are dedicated for program stores. Counter 41 is used for initializing tag ram 32, e.g. in the MEM TO CACHE state. Figure 9 further illustrates how these registers are interconnected, and how the various data paths may be multiplexed to implement the functionality described in conjunction with Figure 7.

[0049] Although the invention has been described herein with reference to a specific processor architecture, it is recognized that one of ordinary skill can readily adapt the described embodiments to operate on other processors, regardless of instruction size, on-chip or off-chip memory size, bus size, or utilization of instruction pipelining. Likewise, nothing in this description should be seen as limiting the possible memory modes of a processor employing a user-configurable memory. For instance, other modes such as explicit boot modes, other known caching modes, and partitioned on-chip modes (multiple cache or part-mapped/part-cache) may be implemented using this disclosure. And although the preferred embodiments have been described using a specific controller design, hose of ordinary skill will recognize upon reading this disclosure that the basic idea of a configurable on-chip memory may be logically implemented in many equivalent designs. Other obvious modifications will be apparent to those of ordinary skill in the art upon reading this disclosure.

[0050] The scope of the present disclosure includes any novel feature or combination of features disclosed therein either explicitly or implicitly or any generalisation thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed by the present invention. The applicant hereby gives notice that new claims may be formulated to such features during the prosecution of this application or of any such further application derived therefrom. In particular, with reference to the appended claims, features from dependent claims may be combined with those of the independent claims and features from respective independent claims may be combined in any appropriate manner and not merely in the specific combinations enumerated in the claims.

Claims

1. A microprocessor comprising:

   a central processing unit; and
   an on-chip memory for storing instructions executable on said central processing unit, said on-chip memory system having a plurality of selectable configurations.

2. The microprocessor of Claim 1, wherein said plurality of selectable configurations comprise both a memory map configuration and a cache configuration.

3. The microprocessor of Claim 1 or Claim 2, wherein said selectable configurations are selectable only during microprocessor boot and reset operations.

4. The microprocessor of Claim 1 or Claim 2, wherein said selectable configurations are selectable by the central processing unit during microprocessor operation.

5. The microprocessor of any of Claims 1 to 4, wherein said on-chip memory comprises a program memory controller arranged for communication with said on-chip memory .

6. The microprocessor of Claim 5, wherein said on-chip memory has a fixed configuration and said program memory controller has a plurality of operational modes, and wherein selection of one of said selectable configurations of said on-chip memory comprises selecting the operational mode of said program memory controller.

7. The microprocessor of Claim 6, wherein said operational modes comprise a memory map mode and a cache enabled mode.

8. The microprocessor of any of Claims 5 to 7, wherein said on-chip memory further comprises a tag memory array controllable by said program memory controller for storing information pertaining to the contents of said on-chip memory array.

9. A microprocessor comprising:

   a central processing unit;
   an on-chip memory for storing instructions executable on said central processing unit;
   an external memory interface capable of reading from and writing to an off-chip memory instructions executable on said central processing unit; and
   a configurable program memory controller arranged for communication with said central processing unit, said on-chip memory array, and said external memory interface, said configurable program memory controller having a plurality of operating modes, including a first mode in which it uses said on-chip memory as a memory-mapped on-chip memory, and a second mode in which it uses said on-chip memory as a cache on-chip memory.

10. The microprocessor of Claim 9, wherein said microprocessor is arranged such that during operation said central processing unit can request said program memory controller to switch from its current operating mode to a different operating mode, and wherein said program memory controller is capable of reconfiguring itself during microprocessor operation in response to said request.

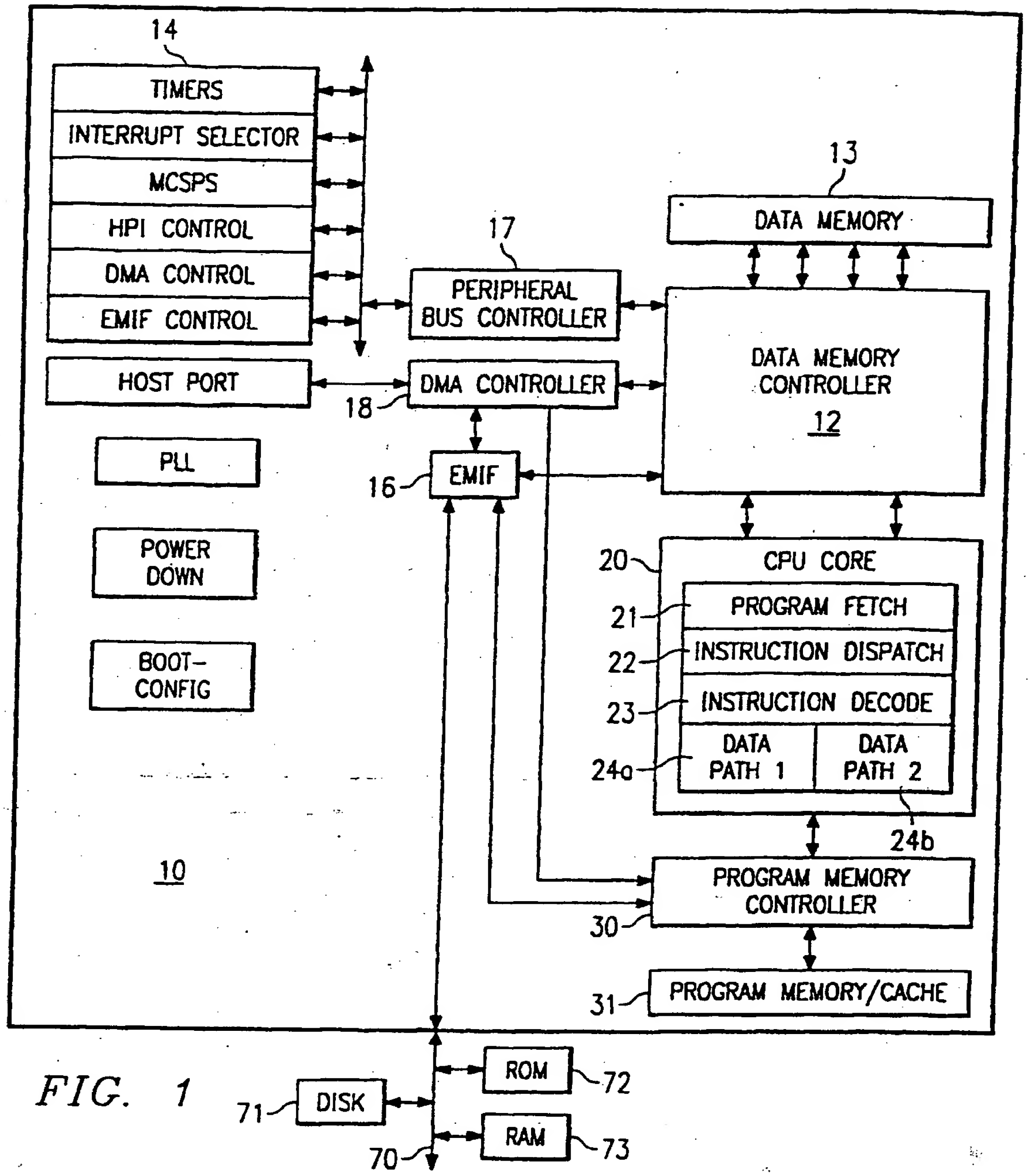

14
TIMERS
INTERRUPT SELECTOR
MCSPS
HPI CONTROL
DMA CONTROL
EMIF CONTROL
HOST PORT
PLL
POWER DOWN
BOOT-CONFIG
10
17
PERIPHERAL BUS CONTROLLER
18
DMA CONTROLLER
16
EMIF
13
DATA MEMORY
DATA MEMORY CONTROLLER
12
20
CPU CORE
21
PROGRAM FETCH
22
INSTRUCTION DISPATCH
23
INSTRUCTION DECODE
24a
DATA PATH 1
DATA PATH 2
24b
PROGRAM MEMORY CONTROLLER
30
31
PROGRAM MEMORY/CACHE
ROM
72
71
DISK
RAM
73
70

FIG. 1

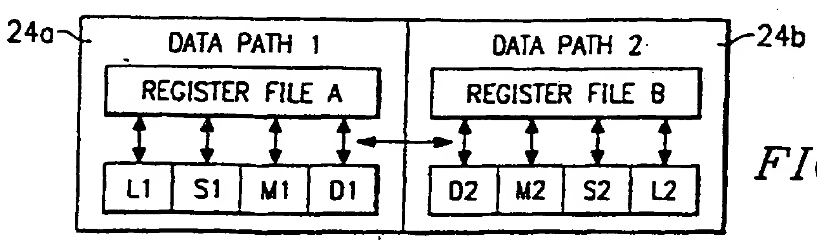

24a
DATA PATH 1
REGISTER FILE A
L1
S1
M1
D1
DATA PATH 2
24b
REGISTER FILE B
D2
M2
S2
L2

FIG. 2

FIG. 3

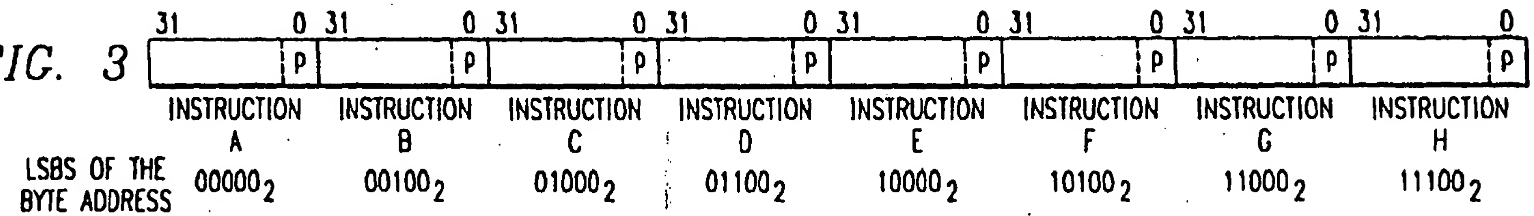
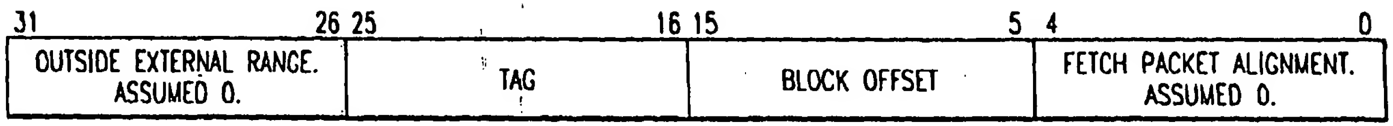


FIG. 5



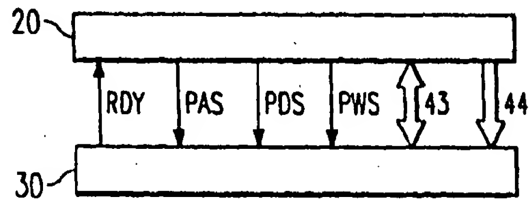


FIG. 6

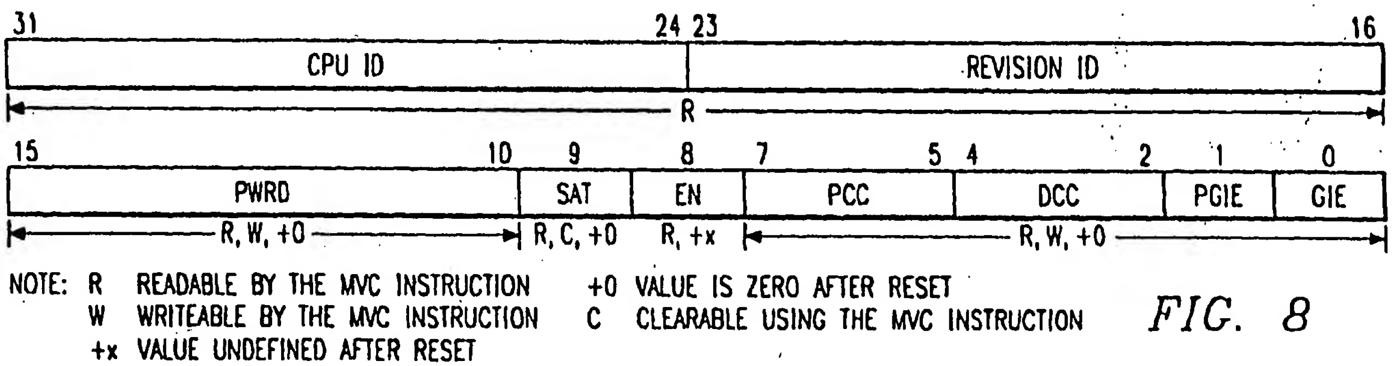


FIG. 8

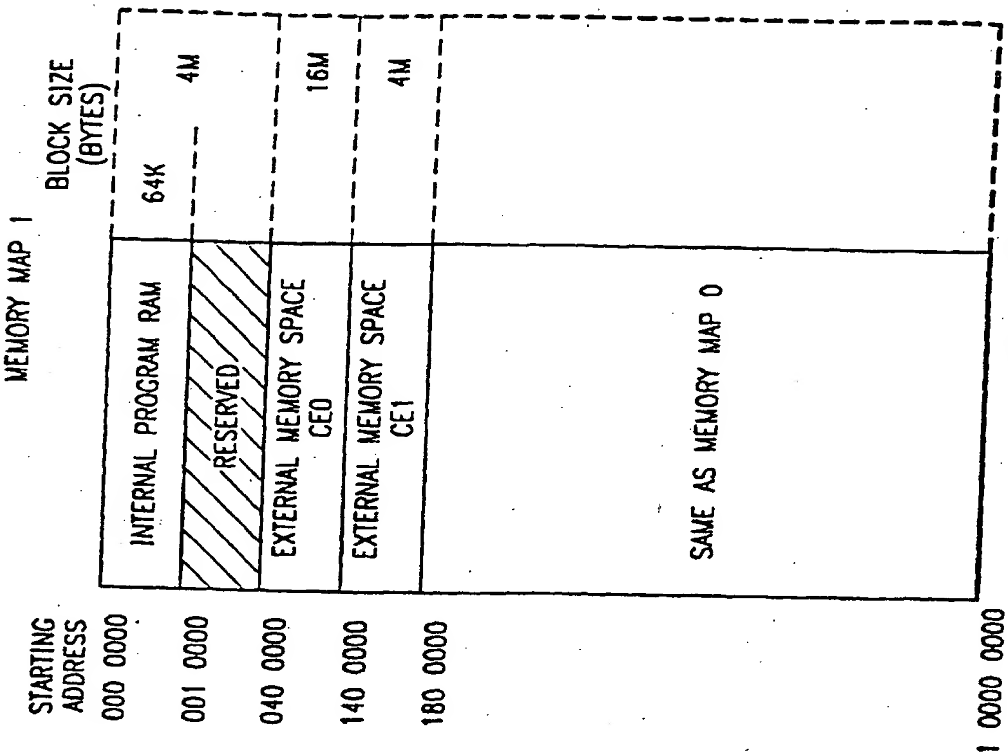

MEMORY MAP 1
STARTING ADDRESS
BLOCK SIZE (BYTES)
000 0000
INTERNAL PROGRAM RAM
64K
001 0000
RESERVED
4M
040 0000
EXTERNAL MEMORY SPACE CE0
16M
140 0000
EXTERNAL MEMORY SPACE CE1
4M
180 0000
SAME AS MEMORY MAP 0
1 0000 0000

FIG. 4b

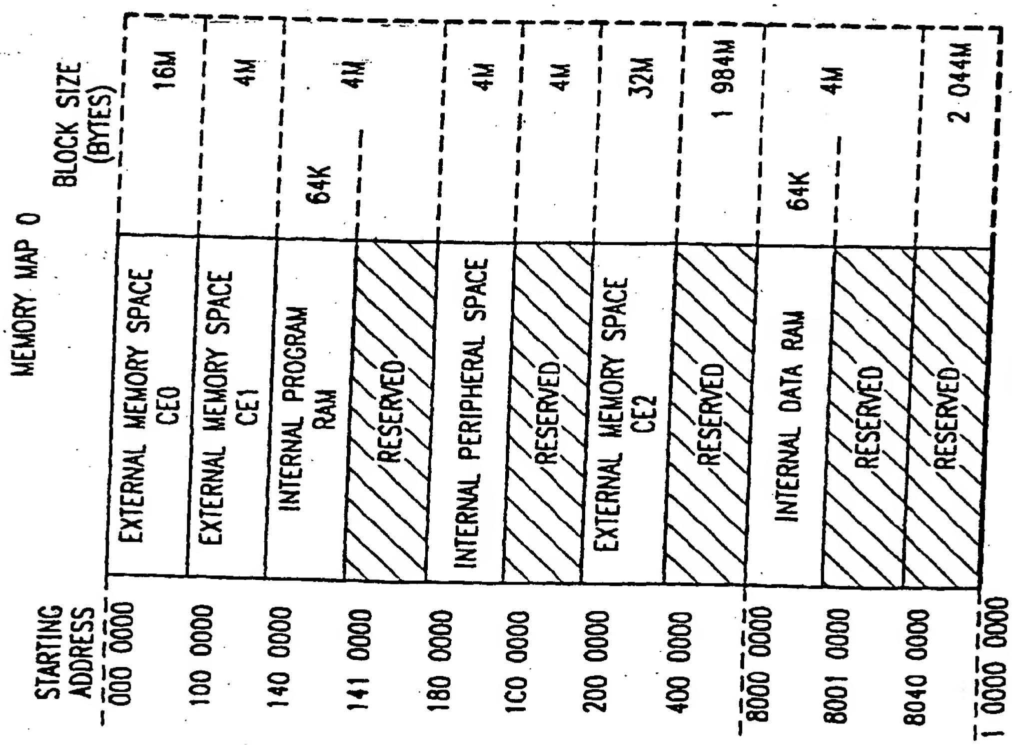

MEMORY MAP 0
STARTING ADDRESS
BLOCK SIZE (BYTES)
000 0000
EXTERNAL MEMORY SPACE CE0
16M
100 0000
EXTERNAL MEMORY SPACE CE1
4M
140 0000
INTERNAL PROGRAM RAM
64K
141 0000
RESERVED
4M
180 0000
INTERNAL PERIPHERAL SPACE
4M
1C0 0000
RESERVED
4M
200 0000
EXTERNAL MEMORY SPACE CE2
32M
400 0000
RESERVED
1 984M
8000 0000
INTERNAL DATA RAM
64K
8001 0000
RESERVED
4M
8040 0000
RESERVED
2 044M
1 0000 0000

FIG. 4a

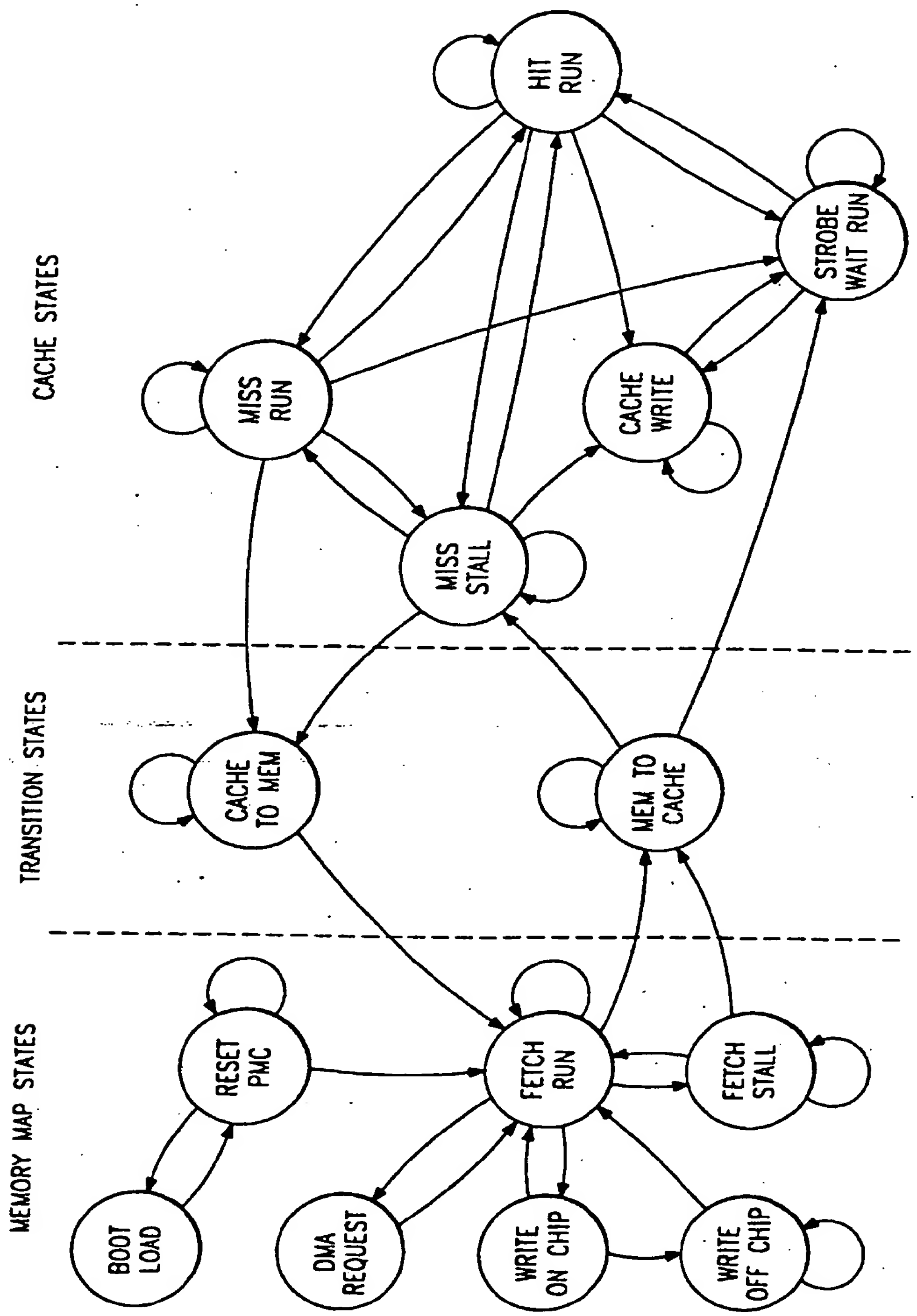

MEMORY MAP STATES
TRANSITION STATES
CACHE STATES
BOOT LOAD
RESET PMC
DMA REQUEST
WRITE ON CHIP
WRITE OFF CHIP
FETCH RUN
FETCH STALL
CACHE TO MEM
MEM TO CACHE
MISS RUN
MISS STALL
HIT RUN
CACHE WRITE
STROBE WAIT RUN

FIG. 7

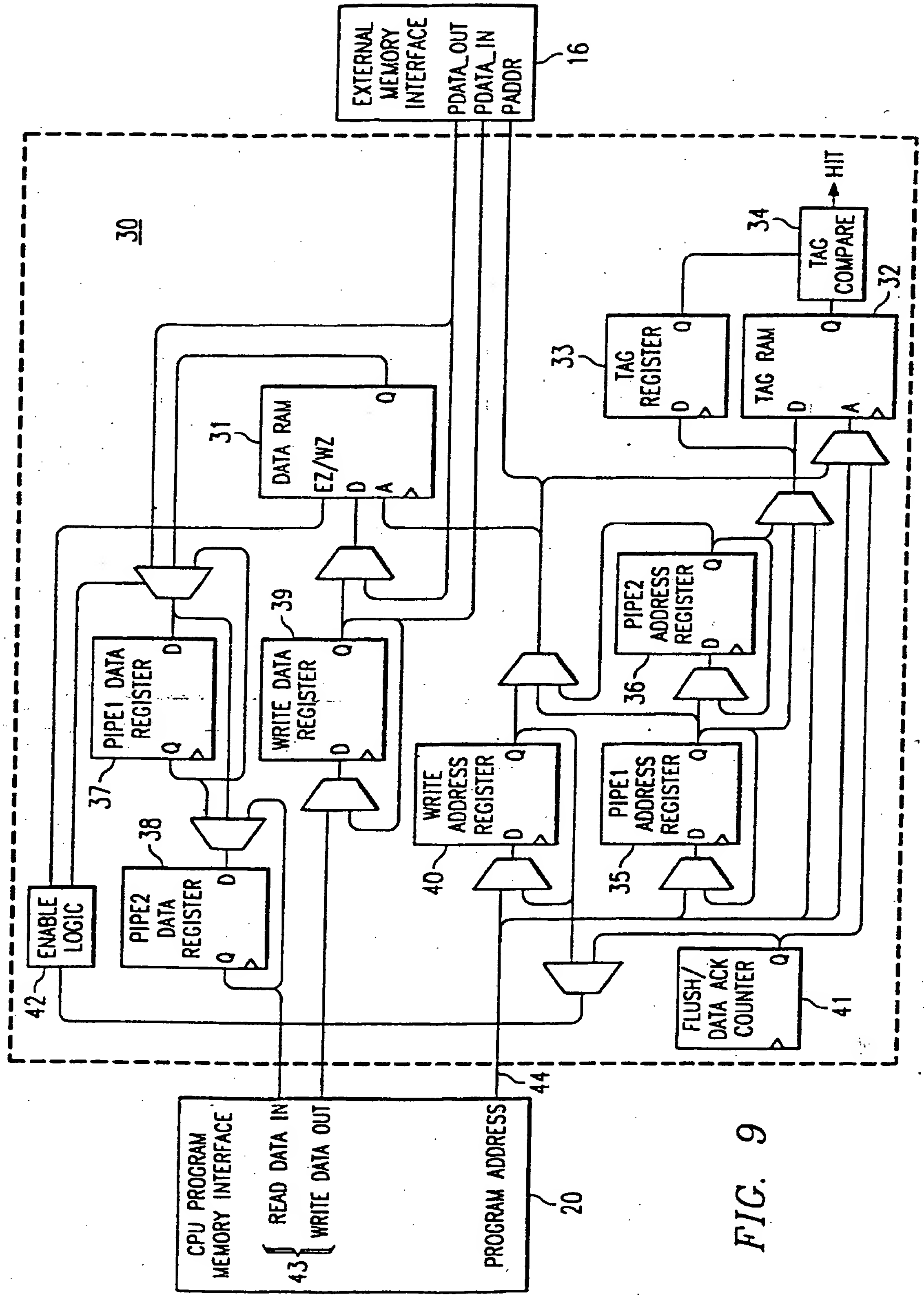

EXTERNAL MEMORY INTERFACE
PDATA_OUT
PDATA_IN
PADDR
16
30
DATA RAM
EZ/WZ
D
A
Q
31
TAG REGISTER
33
TAG RAM
32
TAG COMPARE
34
HIT
PIPE1 DATA REGISTER
37
PIPE2 DATA REGISTER
38
WRITE DATA REGISTER
39
WRITE ADDRESS REGISTER
40
PIPE1 ADDRESS REGISTER
35
PIPE2 ADDRESS REGISTER
36
FLUSH/ DATA ACK COUNTER
41
ENABLE LOGIC
42
44
CPU PROGRAM MEMORY INTERFACE
READ DATA IN
WRITE DATA OUT
43
PROGRAM ADDRESS
20

FIG. 9

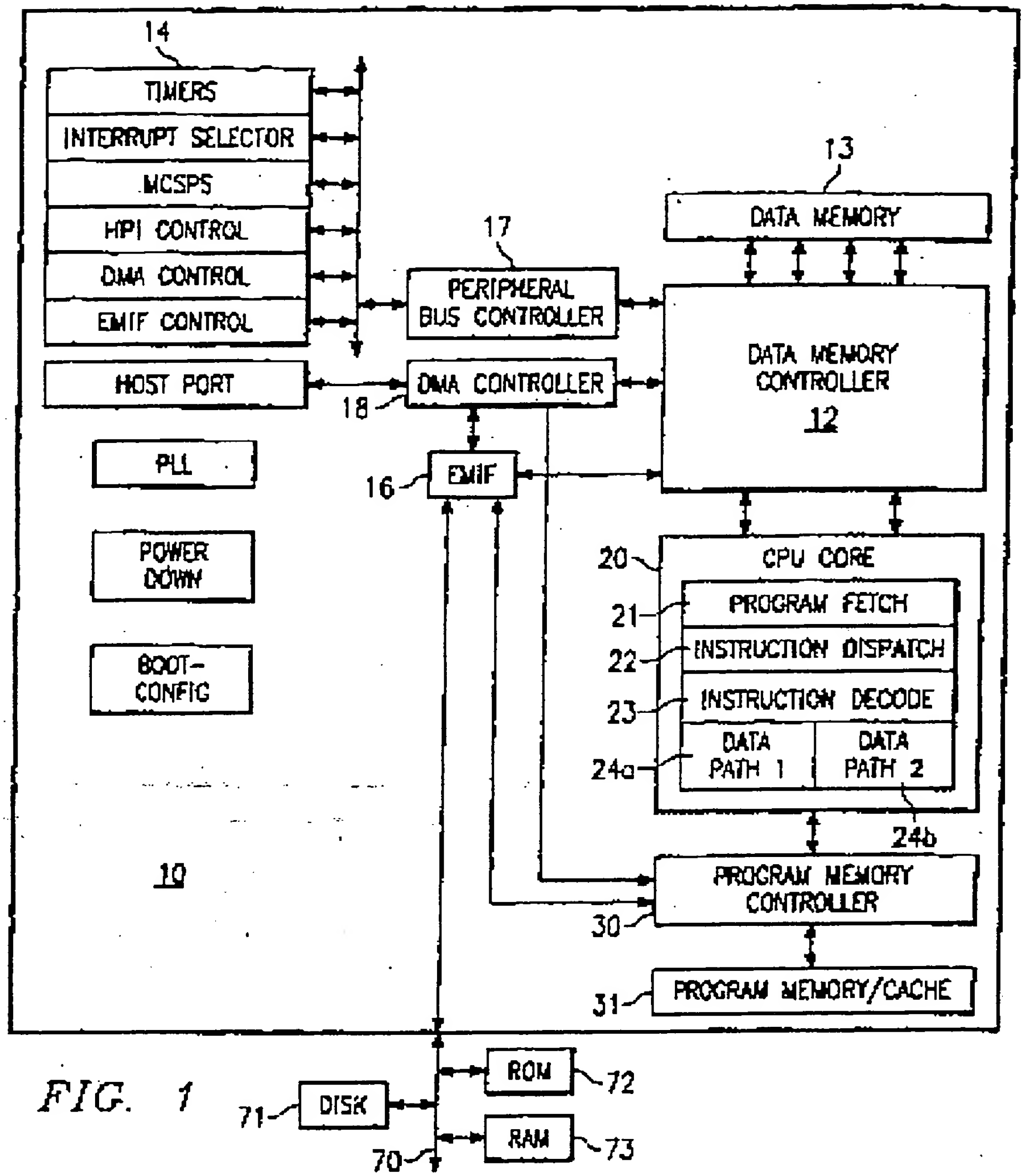

14
TIMERS
INTERRUPT SELECTOR
MCSPS
HPI CONTROL
DMA CONTROL
EMIF CONTROL
HOST PORT
17
PERIPHERAL BUS CONTROLLER
13
DATA MEMORY
DATA MEMORY CONTROLLER
12
18
DMA CONTROLLER
PLL
16
EMIF
POWER DOWN
BOOT-CONFIG
20
CPU CORE
21
PROGRAM FETCH
22
INSTRUCTION DISPATCH
23
INSTRUCTION DECODE
24a
DATA PATH 1
DATA PATH 2
24b
10
PROGRAM MEMORY CONTROLLER
30
31
PROGRAM MEMORY/CACHE
ROM
72
71
DISK
RAM
73
70

FIG. 1

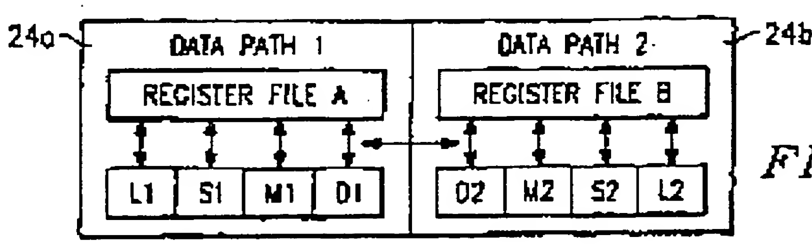

24a
DATA PATH 1
DATA PATH 2
24b
REGISTER FILE A
REGISTER FILE B
L1
S1
M1
D1
D2
M2
S2
L2

FIG. 2

FIG. 3

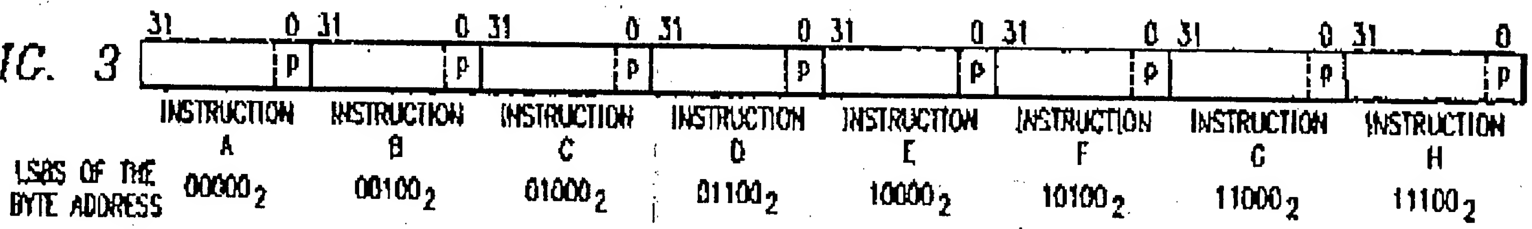
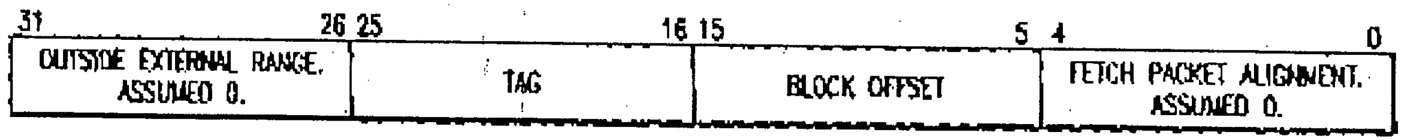


FIG. 5



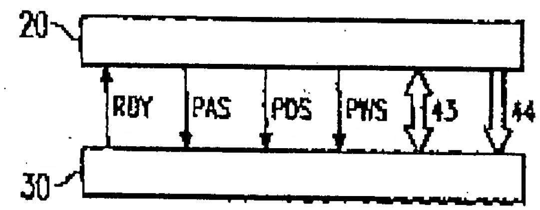
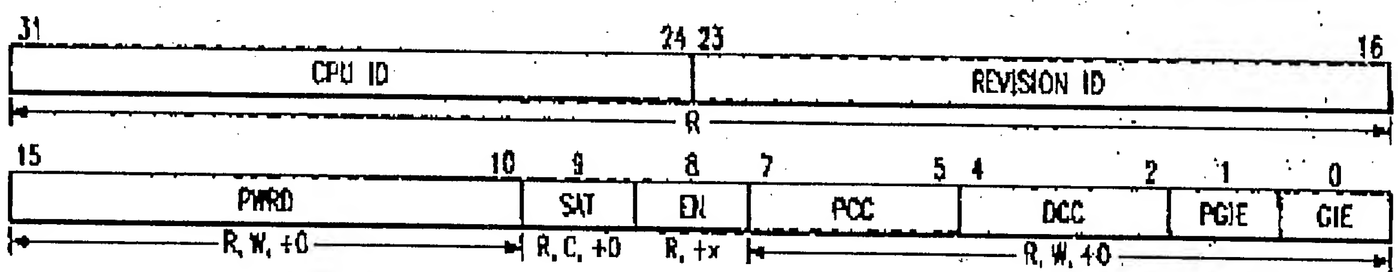


FIG. 6



NOTE: R READABLE BY THE MVC INSTRUCTION
W WRITEABLE BY THE MVC INSTRUCTION
+x VALUE UNDEFINED AFTER RESET
+0 VALUE IS ZERO AFTER RESET
C CLEARABLE USING THE MVC INSTRUCTION

FIG. 8

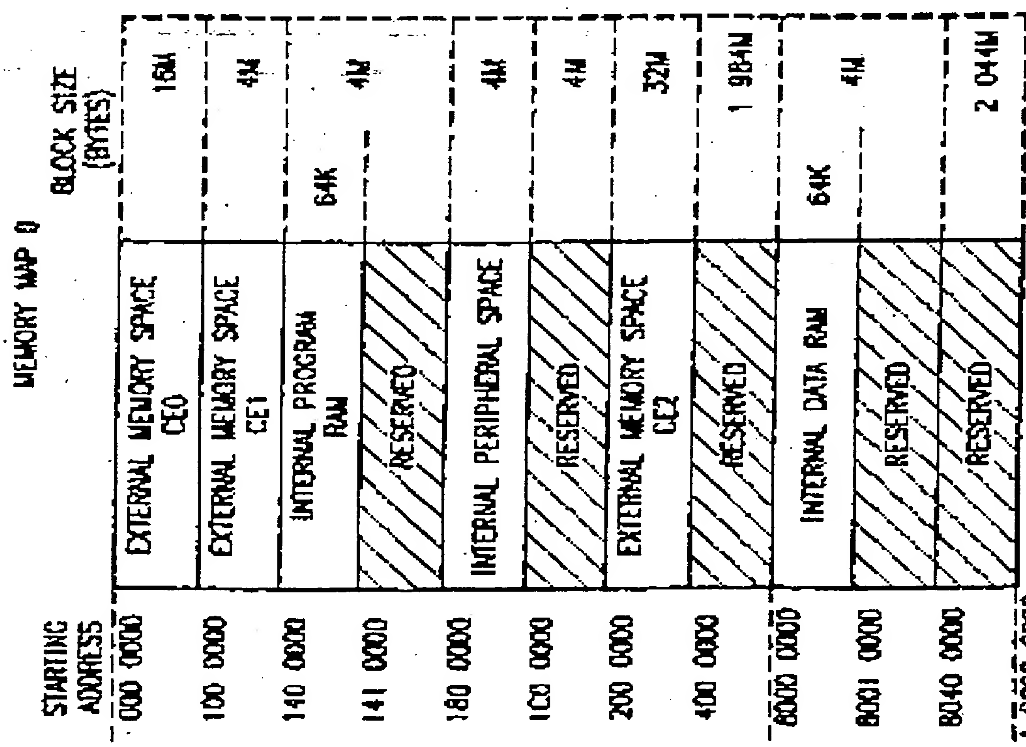

MEMORY MAP 0
STARTING ADDRESS
BLOCK SIZE (BYTES)
000 0000
EXTERNAL MEMORY SPACE CE0
16M
100 0000
EXTERNAL MEMORY SPACE CE1
4M
140 0000
INTERNAL PROGRAM RAM
64K
141 0000
RESERVED
4M
180 0000
INTERNAL PERIPHERAL SPACE
4M
1C0 0000
RESERVED
4M
200 0000
EXTERNAL MEMORY SPACE CE2
32M
400 0000
RESERVED
1 984M
8000 0000
INTERNAL DATA RAM
64K
8001 0000
RESERVED
4M
8040 0000
RESERVED
2 044M
1 0000 0000

FIG. 4a

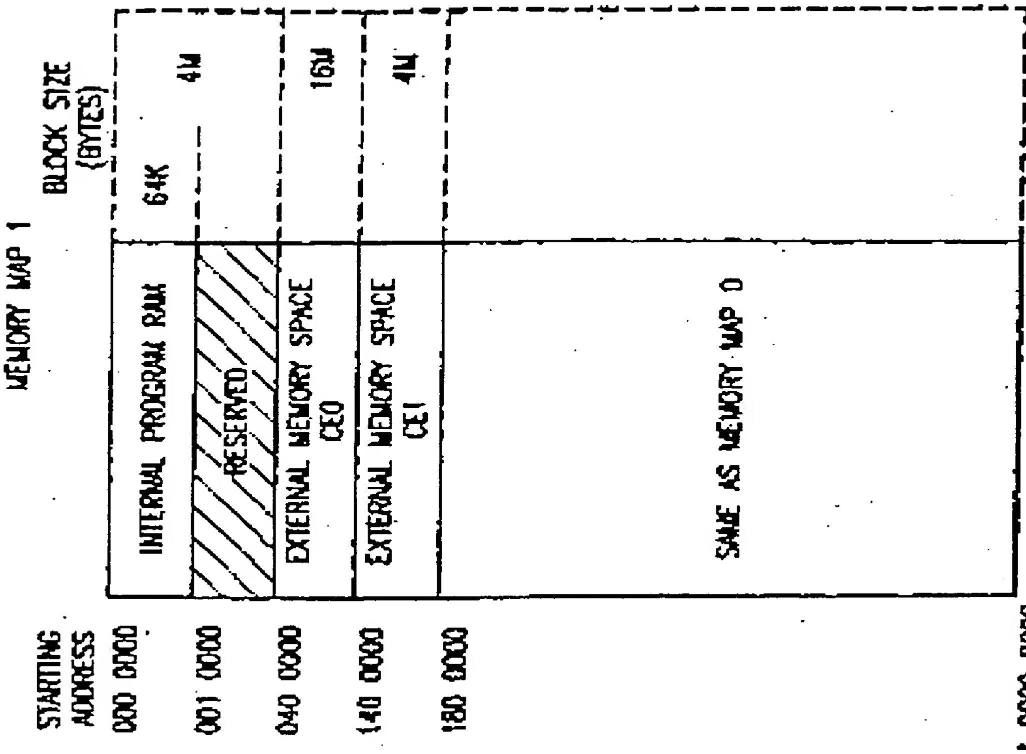

MEMORY MAP 1
STARTING ADDRESS
BLOCK SIZE (BYTES)
000 0000
INTERNAL PROGRAM RAM
64K
001 0000
RESERVED
4M
040 0000
EXTERNAL MEMORY SPACE CE0
16M
140 0000
EXTERNAL MEMORY SPACE CE1
4M
180 0000
SAME AS MEMORY MAP 0
1 0000 0000

FIG. 4b

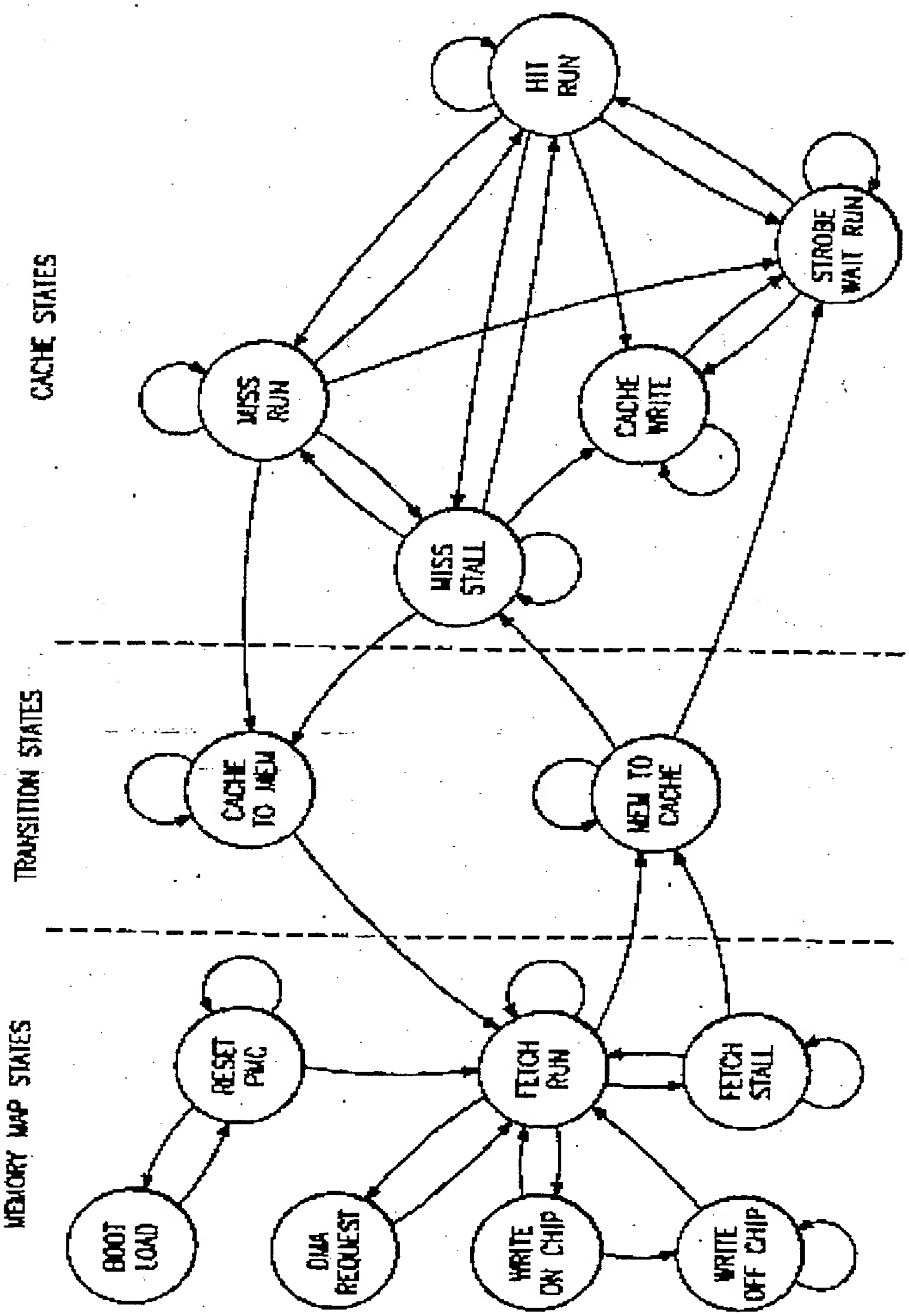

MEMORY MAP STATES
TRANSITION STATES
CACHE STATES
HIT RUN
MISS RUN
MISS STALL
CACHE WRITE
STROBE WAIT RUN
CACHE TO MEM
MEM TO CACHE
RESET PMC
BOOT LOAD
DMA REQUEST
FETCH RUN
FETCH STALL
WRITE ON CHIP
WRITE OFF CHIP

FIG. 7

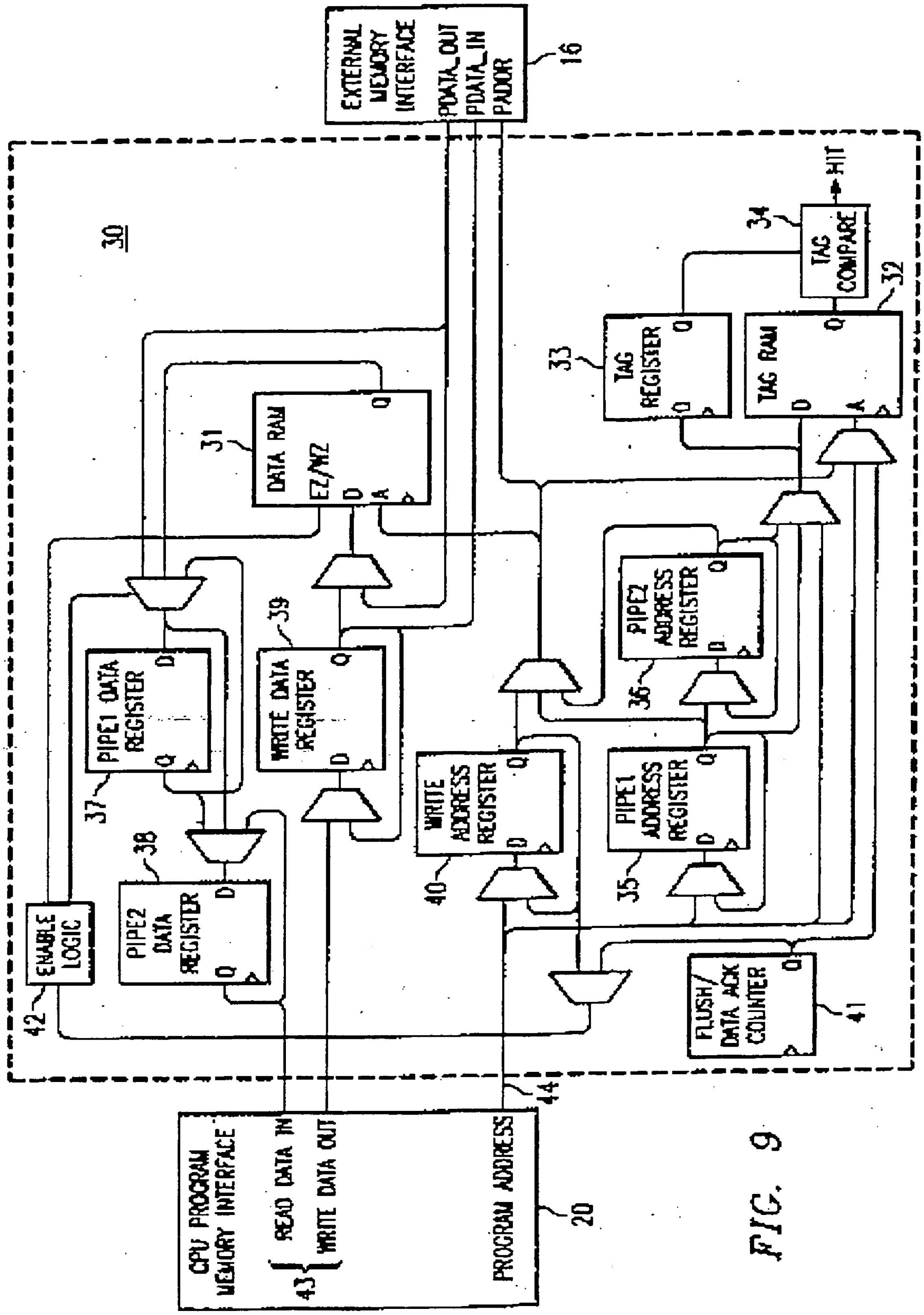

EXTERNAL MEMORY INTERFACE
PDATA_OUT
PDATA_IN
PADDR
16
30
HIT
34
TAG COMPARE
32
33
TAG REGISTER
TAG RAM
31
DATA RAM
EZ/NZ
39
PIPE1 DATA REGISTER
WRITE DATA REGISTER
37
38
PIPE2 DATA REGISTER
PIPE2 ADDRESS REGISTER
36
WRITE ADDRESS REGISTER
40
PIPE1 ADDRESS REGISTER
35
ENABLE LOGIC
42
FLUSH/ DATA ACK COUNTER
41
44
CPU PROGRAM MEMORY INTERFACE
READ DATA IN
WRITE DATA OUT
43
PROGRAM ADDRESS
20

FIG. 9